Itanium 2 MicroArchitecture and Performance Tuning

Jin Guojun

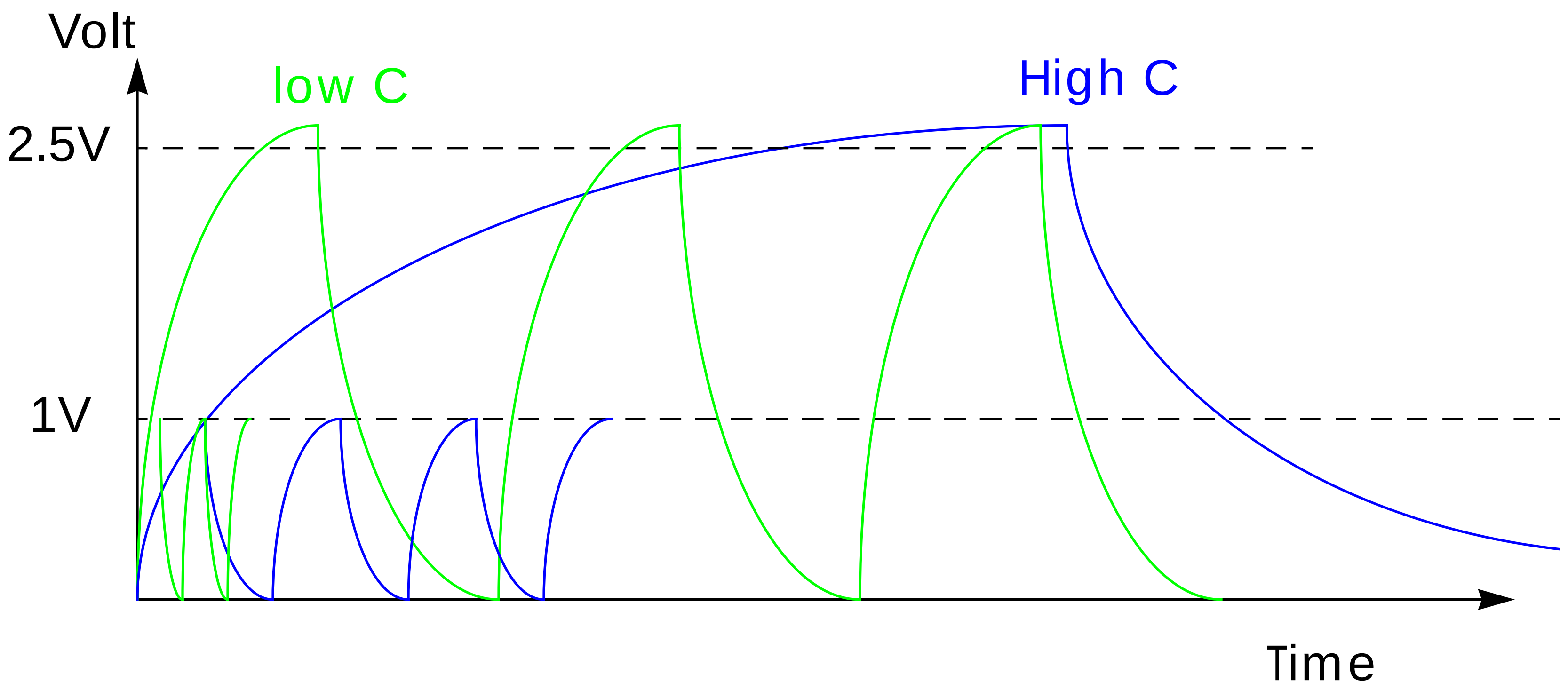
Lawrence Berkeley National Laboratory

Apr. 30, 2004

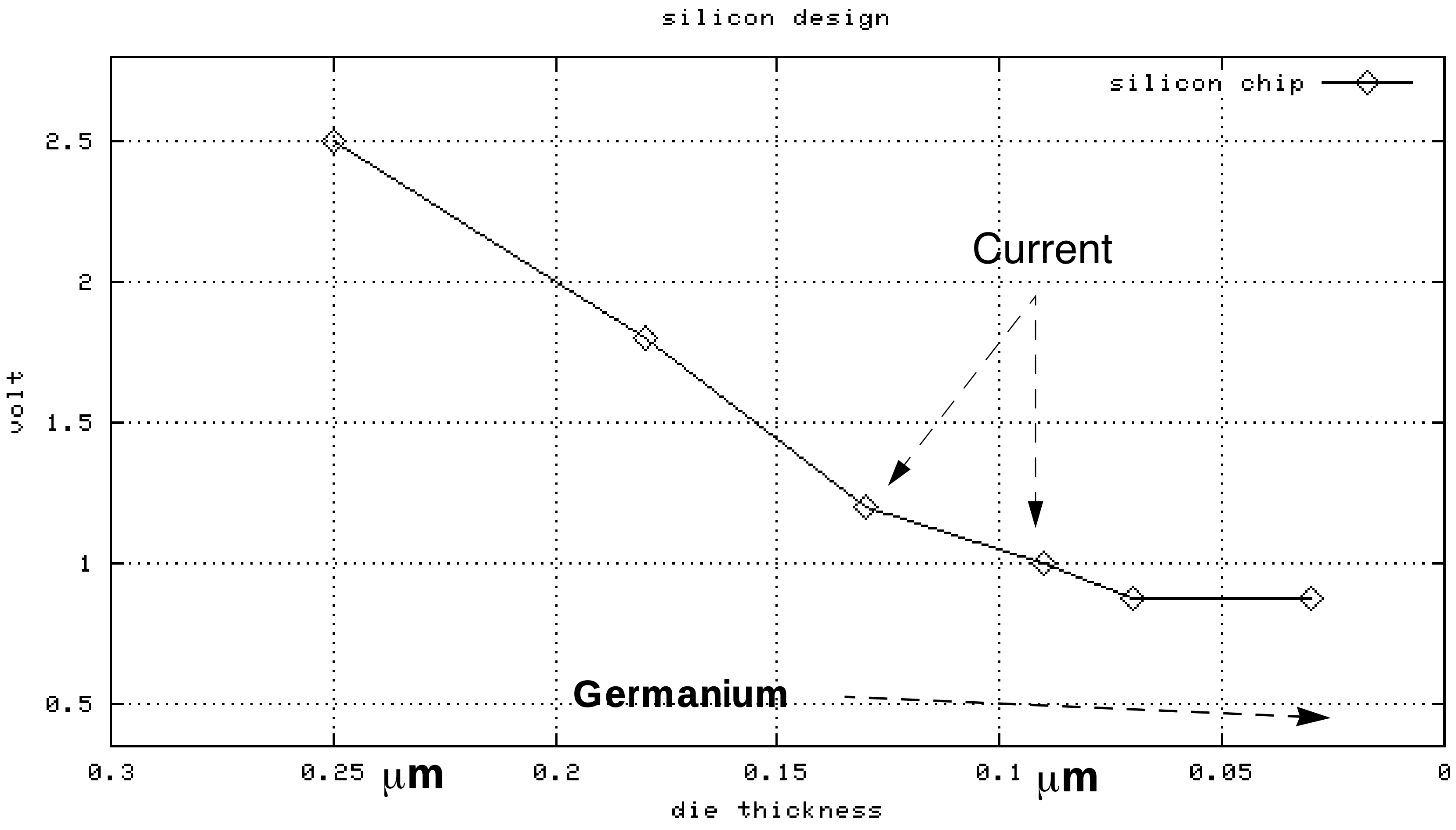
Contents

Design and Determine CPU Speed

The thinner the die is, the lower the voltage and capacitor will be



Chip Design



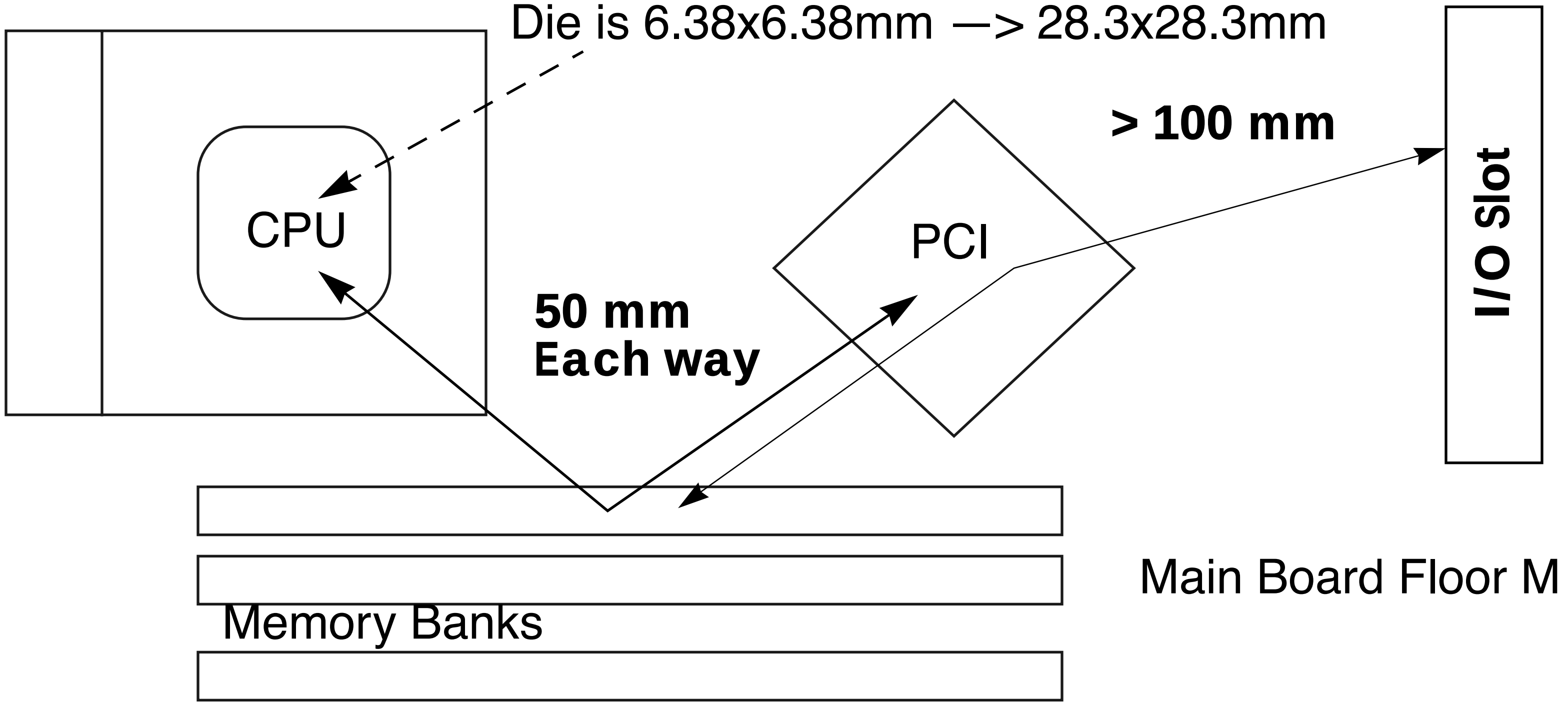
Further lower the voltage, static power will be higher than active power

How Far the Silicon can GO

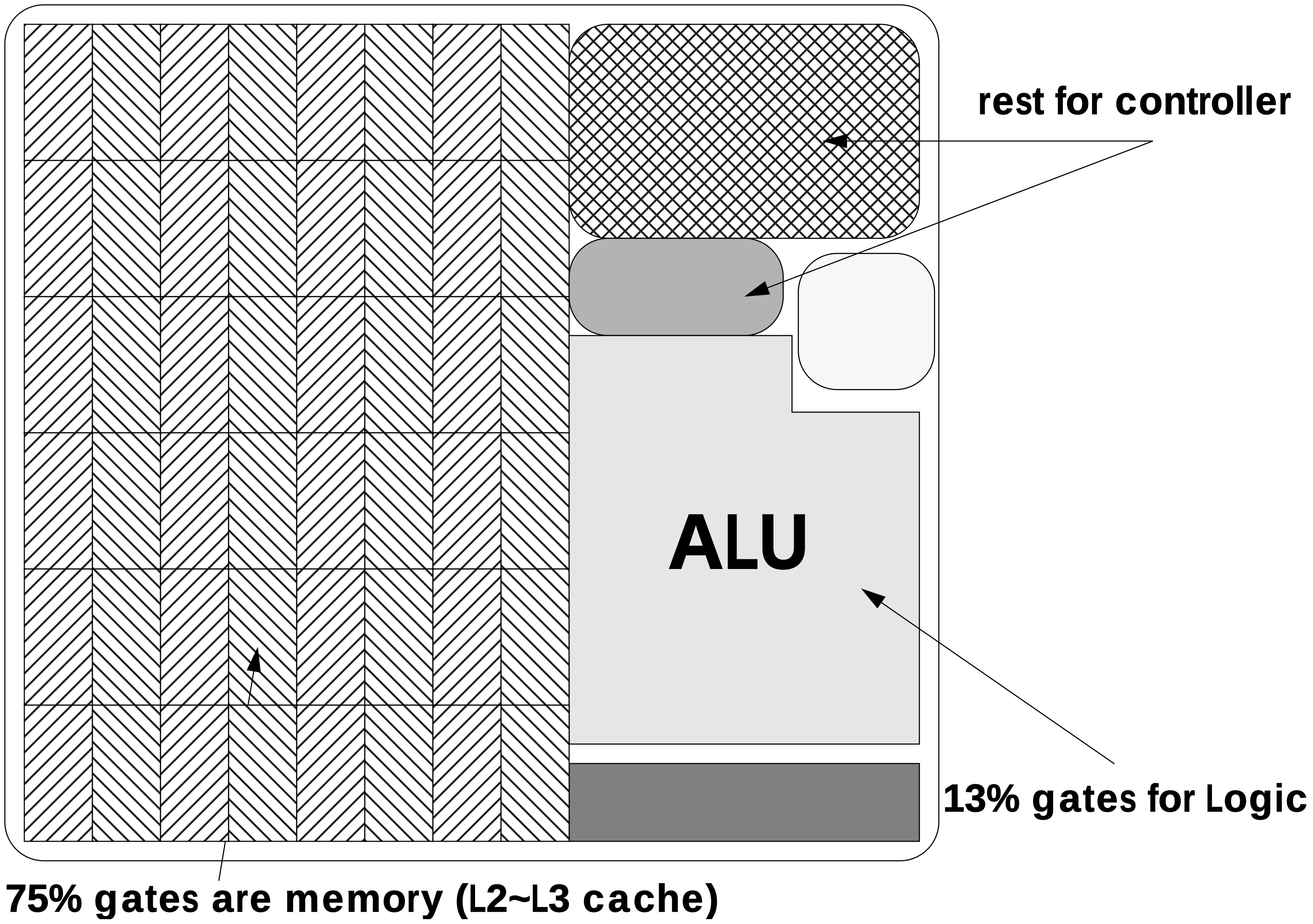
(1) Light speed is 300,000 Km/sec = 300mm/ns

10GHz clock —> 0.1 ns per clock cycle, light can travel 30mm per clock cycle

With 100GHz clock, light can travel only 3 mm per clock cycle



CPU layout

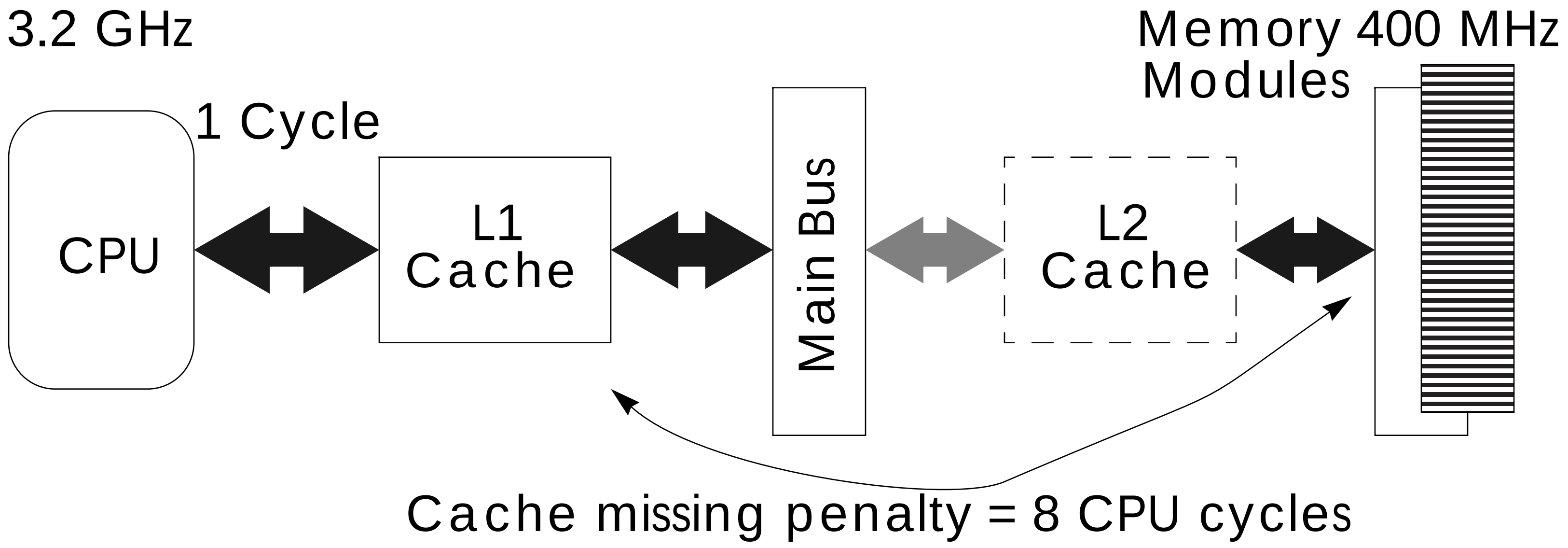

rest for controller
ALU
13% gates for Logic
75% gates are memory (L2~L3 cache)

Memory Bandwidth

$$\text{MemoryBandwidth} \neq \text{BusClockRate} \times \text{BusWidth}$$

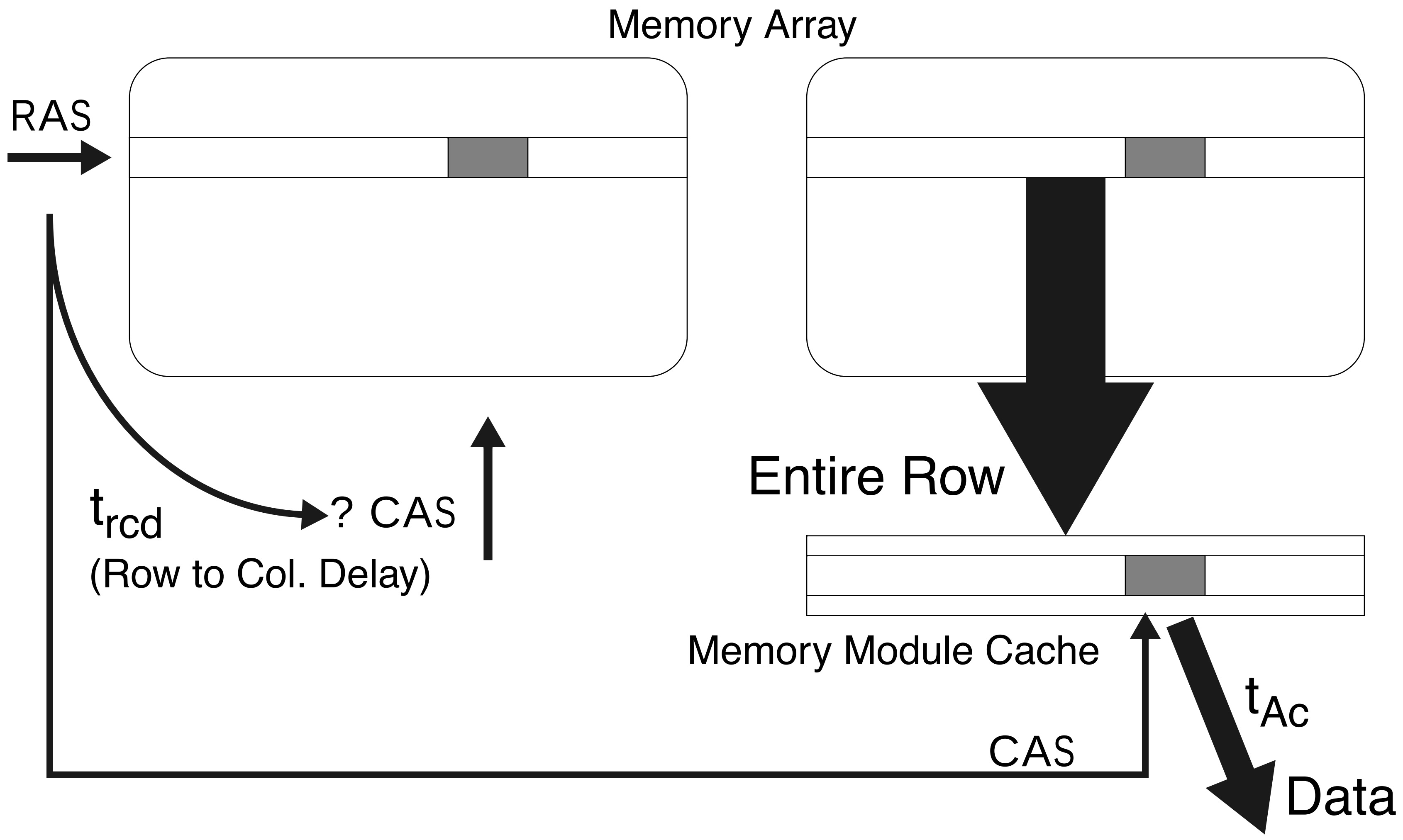
64-bit (8 bytes) 400 MHz memory system does **NOT** produce 3.2 GB/s memory bandwidth:

This is because of cache, memory and I/O controllers

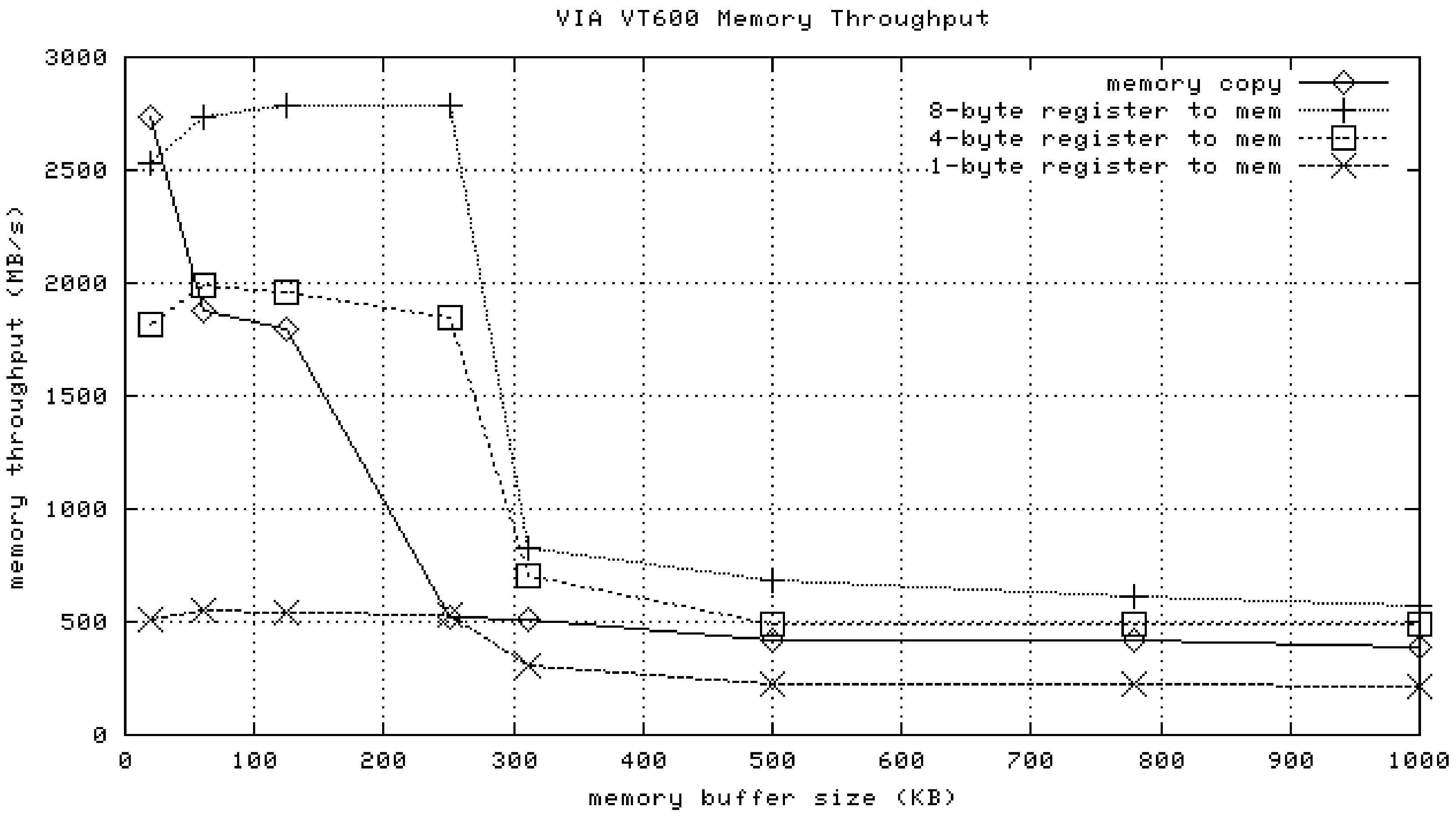


Cache missing penalty = 8 CPU cycles

Memory Subsystem



Memory Bandwidth



Design Issues

- Hardware Design — microarchitecture
    - EPIC instruction
    - pipeline
    - L1-L3 cache
    - effort on compiler and hardware designs
- Software Design Issues —
    - Protocol
        - data structure
        - subroutine argument and local data declaration
    - Memory organization
- Vtune — Performance analyser

Architecture Strategy

- Parallelism
  - Three instructions per bundle and two bundles per clock
  - Allow compiler to exploit parallelism by eliminating static scheduling
- Branches Improvement
  - Eliminate branches with predication
  - Reduce the number of branch mispredicts by using branch hints
  - Execute more than one branch per clock by multiway branch
- Register stack
  - Reduce procedure call/return overhead
- Level 3 cache
  - Increase relatively to processor cycle time

Instruction

- Format
- Group
- Parallelism
- Predication
- Pipeline

Instruction Format

[**qp**] *mnemonic*[.comp] *dest* = *srcs*

qp = qualifying predicate — 1 bit special register (64)
When ***qp*** is missing, it means true.

mnemonic = unique name identifying the instruction
comp = one or more completers
dest = destination operand(s)
srcs = source operand(s)

Example:

```
(p10) ld4.s r31 = [r3]
```

Instruction Group

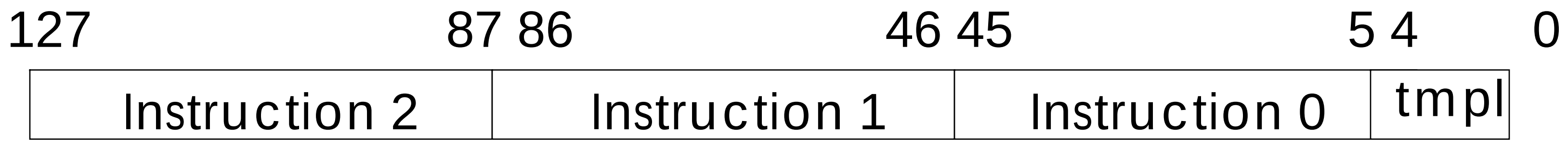
Contiguous instructions that do not have dependencies
Terminated by an instruction group boundary “;;”

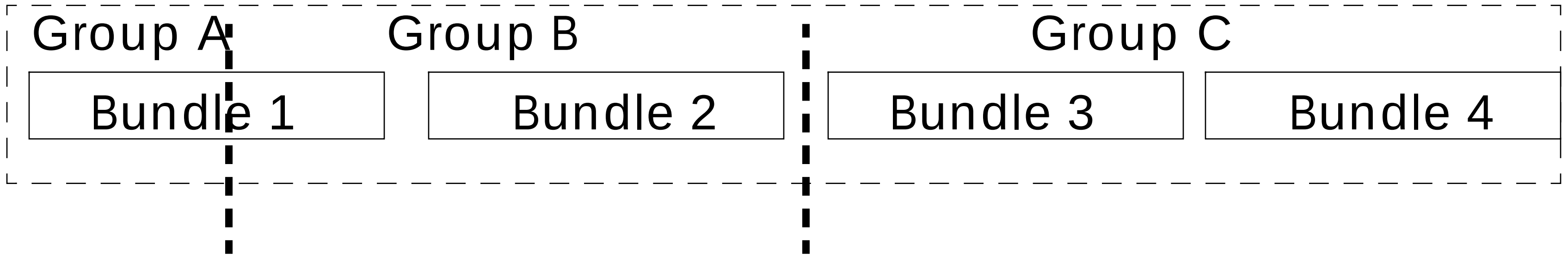
Example:

```
add     r31 = r5, r6 ;;     Group A

mov     r4 = r31
add     r2 = r8, r9 ;;       Group B

mov     r7 = r2
mov     r15 = r27
mov     r16 = r28
mov     r17 = r29
add     r3 = r11, r20 ;;     Group C

mov     r10 = r3             Group D
```

Parallelism

50-bit physical addressing, 64-bit virtual addressing
41-bit instruction
6 integer units
2 bundles per clock
Bundle — 128 bits





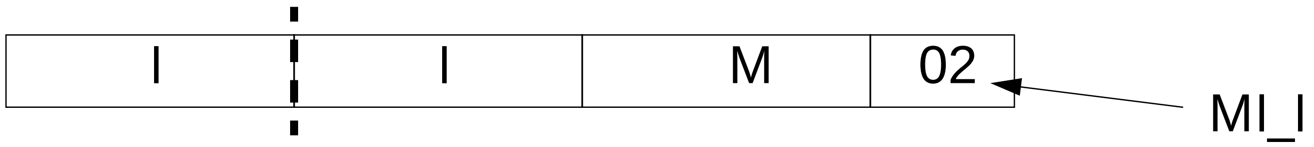
Bundle Cont'd

Templates:

MII, MLX, MMI, MFI, MMF, MIB, MBB, BBB, MMB, MFB

MI_I, M_MI
MII_, MLX_, MMI_, etc.

M = Memory
I = Integer
A = Memory/Integer
F = Floating-point
L+X = Extended (Immediate data)
B = Branch
Example:



Registers

- 128 general registers (64 bits + 1)

    each GR is associated with an 1-bit CR for communication. e.g., if memory is available, etc.

    0-31 is the main register frame
    32-127 is register sliding window

    To avoid window overflow, deep procedure calls is dis-encouraged

- 8 branch registers (64 bits)
- 128 floating point registers (82 bits)
- 64 predicate registers (1 bit)
- Instruction Pointer (IP) (64 bits)
- Application registers
- Performance monitoring data registers
- Processor identification registers (CPUID)

Branch and Pipeline

[qp] mnemonic[.comp] dest=srcs

- If [qp] is missing, it means true.

```
if (condition is true)   ✓
   here
else
   somewhere-else

If (condition is False)   ✘
   somewhere-else
else
   here
```

Branch and Predication

Non-predicated

```
if (a > b)   ++c
else   d = d*e + f
```

Predicated

```
pT, pF = compare (a>b)
if (pT)   ++c
if (pF)   d = d*e + f
```

Conditions and branches in a loop are hinted by predication registers (via %).

Eight-Stage Pipeline

- Two-stage front end gets and formats instructions from L1-I cache to the instruction streaming buffer
  - Front end loads pipeline instruction buffer, which stages instructions for back end
- Six-stage pipeline back end
  - Expands the templates (EXP)
  - Prepares registers for access by the instructions (REN)
  - Loads data from registers to functional units (REG)
  - EXE stage invokes instructions and routes output from single cycle ALU's back to REG stage as needed
  - DET stage detects micropipeline stalls, exceptions and branch mispredictions and flushes the pipeline
  - WRB stage writes output of functional units to registers

This means that mis-predication will have at least 8 cycles penalty.

Cache

System Bus — 128 bits wide
200MHz / 400 MTx/s -> 6.4GB

L1 — 2X16KB   1 clock latency (64B lines — 8 words)
L2 — 256K   5 clock latency (128B lines — 16 words) [32GB MBW]
L3 — 6MB On die 14~17 clock latency (48GB/s bandwidth to L2 ???)

2GHz CPU <--> Memory clock ratio = 10
All cache missing, what is the memory delay penalty?

Do not allocate (static or dynamic) data array at multiple of 256Bytes!

		line	1	of	8					line	2	of	8		
		line	3	of	8					line	4	of	8		
		line	5	of	8					line	6	of	8		
		line	7	of	8					line	8	of	8		
b0															b15

L2 Unified Cache Bank: 16 banks cover 256 Bytes = 2 cache lines

Cache Access

- L2 data access controlled by 32 entry queue (OzQ — register line) and allows out of order data return
  - FP data loaded to FP register directly from L2
- Minimum integer latency is 5 cycles
- Minimum floating point latency is 6 cycles
- Latency is increased by
  - Cache miss
  - Bank conflicts cause OzQ cancels (measured to add 6 cycles)
  - Multiple misses and misses to lines being updated will cause OzQ recirculates (measured to add 16 cycles)
  - only one data access is escalated to L3 and the system bus, the others recirculate

Microbanchmarks

Example: Simple microbanchmark to measure cache latencies

- Time two simple loops with and without loads and divide difference by the iteration to yield the latency

Example:

```
cache access latency                    baseline
ld    r29 = [r34], 128;;                nop.m 0;;
mov r28 = r29                           mov r28 = r29
br.ctop.sptk top_of_loop                br.ctop.sptk top_of_loop

   average 7.12 cycles                     average 3.02 cycles
```

Bandwidth Considerations

- Maximum front side bus bandwidth is 6.4 GB/s

- Program bandwidth (BW) required by loop

  BW = Lines_per_iter x 128 bytes x 1GHz / cycles_per_iter

  - Lines_per_iter mush include read and 2 * write output lines

- Setting the program BW < 6.4 GB/s determines minimum cycles per iteration

  - Cycles_per_iter > Lines_per_iter x 128 / 6.4

Data Structure

```
struct my_data   {
    int      md_I1,
             md_I2,
             md_I3;
    short    md_S1;
    char     md_C0,
             md_Cpad[1];               // reserved space
    char     *md_Cp;
    double md_D0;
};

struct m_data   {
    int      md_I1;
    short    md_S1;                    // 2 bytes pad are inserted
    char     *md_Cp,
             md_C0;                    // 7 bytes pad are inserted
    double md_D0;
    char     md_C1;                    // 7 bytes tailing
} packets[8];                          // total 320 bytes, not 144 (packed)
```

Compiler

Due to limited die's space, hardware becomes simplified and dumb. This requires compiler to be more smart, and human error will affect performance.

Intel 8.x Compilers:
Optimization for Linux on Itanium Processor Family Systems

Optimization

POV-Ray software	run time
1. make usegcc —	32s
2. make useicc —	26s
3. make useicc CF=”-O3” LF=”-O3” —	23s (HLO)
4. make useicc CF=”-tpp#” LF=”-tpp#” —	25s (HLO, # = 1,2)
5. make useicc CF=”-ipo” LF=”-ipo” —	24s (Interprocedural opt)
6. make useicc CF=”-prof_gen” LF=”-prof_gen” —	23s
7. make useicc CF=”-O3 -tpp2 -ipo -prof_use” LF=”...” —	17s

VTune ™ Performance Analyzer

Native Performance Analysis

- Intel® IA-32 Processors
  - MS Windows (GUI + command line)
  - Linux — command line only
- Itanium Family Processors
  - Same as above
- For specific operating systems versions, see the release notes

Used for understanding where is the bottleneck of your programs